# Maven vs Gradle

|  |  |  |
| --- | --- | --- |
| **Feature** | **Maven** | **Gradle** |
| **Build Script Language** | XML | Groovy or Kotlin DSL |
| **Flexibility/Extensibility** | Convention-over-configuration | Highly flexible and extensible |
| **Performance** | Strict lifecycle phases, potential for slower builds | Incremental builds, faster for large projects |
| **Dependency Management** | Centralized repository system, POM file | Centralized repository, more flexible expressions |
| **Plugin Ecosystem** | Robust plugin ecosystem, XML configurations | Extensive plugin system, Groovy/Kotlin scripts |
| **Adoption/Community** | Widely adopted in Java community | Growing popularity, widely used in Android projects and enterprises |

Both Gradle and Maven are popular build automation and project management tools used in the Java ecosystem (and beyond). While they share some similarities, there are key differences between the two:

1. **Build Scripting Language:**
   * **Maven:** Maven uses XML for its build scripts. The configuration is declarative, and users define the project structure, dependencies, and build phases in XML format.
   * **Gradle:** Gradle uses Groovy or Kotlin DSL for build scripts. This allows for a more expressive, concise, and readable syntax compared to XML. Users can also write custom scripts to perform various build tasks.
2. **Flexibility and Extensibility:**
   * **Maven:** Maven follows a convention-over-configuration approach. While this can simplify the build process, it may be less flexible in certain scenarios where custom configurations are needed.
   * **Gradle:** Gradle is known for its flexibility and extensibility. It allows developers to customize and extend the build process easily. This flexibility is especially beneficial for complex projects or unique requirements.
3. **Performance:**
   * **Maven:** Maven relies on a predefined set of lifecycle phases, and it can be perceived as less performant for large projects due to its strict order of execution.
   * **Gradle:** Gradle offers incremental builds, meaning it only builds the parts of the project that have changed since the last build. This can lead to faster build times, particularly for larger projects.
4. **Dependency Management:**
   * **Maven:** Maven uses a centralized repository system to manage dependencies. It relies on a Project Object Model (POM) file to define project information and dependencies.
   * **Gradle:** Gradle also uses a centralized repository for dependency management. However, its build scripts are more flexible in expressing dependencies, and it supports transitive dependencies more efficiently.
5. **Plugin Ecosystem:**
   * **Maven:** Maven has a robust plugin ecosystem, and many plugins are available for common tasks. However, configuring and customizing plugins may require additional XML configurations.
   * **Gradle:** Gradle plugins are written in Groovy or Kotlin, and they often provide more flexibility and ease of use compared to Maven plugins. Gradle's plugin system is designed to be more developer-friendly.
6. **Adoption and Community:**
   * **Maven:** Maven has been around longer and is widely adopted in the Java community. Many projects and organizations use Maven for its stability and mature ecosystem.
   * **Gradle:** Gradle has gained popularity for its modern approach and flexibility. It is the build tool of choice for various Android projects and is increasingly used in enterprise settings.

# Dependency BOM

## Project Code Management

For a better dependency management among platform projects, first we have to understand how projects code bases are stored.

Project code base management in terms of repositories.

A screenshot of a diagram

Description automatically generated

[[image - 001 - Projects code management - Repositories ]](https://medium.com/burak-tasci/full-stack-monorepo-part-i-go-services-967bb3527bb8)

In a mono-repo pattern its easy to implement common dependency management since child projects can inherit from a parent project as illustrated below.

A screenshot of a computer

Description automatically generated

[[ image - 002 - Dependency management inheritance from parent to child projects ]](https://liviutudor.com/2016/10/27/gradle-multi-module-projects-centralized-configuration/)

RBC digital uses Microservices architecture in Multi repo pattern. Hence if we are to manage common dependencies and versions of all projects we are forced to use an external resource approach.

## Bill-Of-Materials POM : BOM

Maven’s dependency management includes the concept of a bill-of-materials (bom). A bom is a special kind of pom that is used to control the versions of a project’s dependencies and provides a central place to define and update those versions.

[[ Better dependency management for Gradle (spring.io) ]](https://spring.io/blog/2015/02/23/better-dependency-management-for-gradle)

A bill-of-materials POM is a POM that can declare bundles of dependencies that have been tested to work well together. The abundance of artifacts and versions of each can, at times, lead to confusion and needs for trial-and-error mechanisms to determine compatibility and/or right functionality. A bill-of materials POM reduces that overhead. A bill-of-materials POM is a means of multiple inheritance too, since a project POM can import multiple bill-of-material POMs. The bill-of-materials POM is unaware of the child POMs that import it. The child POMs declare the bill-of-materials in the project POM.

A diagram of a project

Description automatically generated

[[ image - 003 - BOM explanation ]](https://cguntur.me/)

## The Java Platform Plugin - Gradle way of implementing BOM

Java platform plugin is available since gradle version 5.2. ( Gradle 5.2 Release Notes )

The Java Platform plugin brings the ability to declare platforms for the Java ecosystem. A platform can be used for different purposes:

* a description of modules which are published together (and for example, share the same version)
* a set of recommended versions for heterogeneous libraries. A typical example includes the [Spring Boot BOM](https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-dependency-management)
* [sharing a set of dependency versions](https://docs.gradle.org/current/userguide/java_platform_plugin.html#sec:java_platform_consumption) between subprojects

A platform is a special kind of software component which doesn’t contain any sources: it is only used to reference other libraries, so that they play well together during dependency resolution.

Platforms can be published as [Gradle Module Metadata](https://github.com/gradle/gradle/blob/master/subprojects/docs/src/docs/design/gradle-module-metadata-latest-specification.md) and [Maven BOMs](https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Dependency_Management).

[[ The Java Platform Plugin (gradle.org) ]](https://docs.gradle.org/current/userguide/java_platform_plugin.html#java_platform_plugin)